Mini Project Report Guidelines for Full Stack Development Course

MIT-WPU
।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Submitted to	Date	Sign
Prof. Sagar Apune	10/11/2025	

Submitted by	Roll No.	PRN
Abhinav Magesh	11	1032203073
Amar Soni	31	1032231364
Shikhar Kshatriya	33	1032231465

Metro Rail Document Management System (DMS) with AI-Driven Ingestion and Summarization

2. Abstract

This report details the design, development, and implementation of a modern, web-based Document Management System (DMS) prototype for a metro rail administration. The project's purpose is to solve the inefficiencies of traditional, siloed document handling by creating a secure, centralized, and intelligent platform. The system is built on a serverless architecture using React for the frontend, Google Firebase (Authentication and Firestore) for the backend, and the Gemini API for generative AI capabilities. Key features include a robust Role-Based Access Control (RBAC) system for three distinct user roles (Admin, Engineer, Vendor), multi-channel document ingestion (including text, file, and image-based OCR), and automatic AI-powered document summarization upon submission. The resulting application is a secure, responsive, and scalable prototype that successfully demonstrates the feasibility of integrating generative AI and serverless technology to streamline critical administrative workflows, reduce manual data entry, and enhance data security.

3. Introduction

3.1 Background and Motivation

- Large-scale infrastructure operations, such as a city's metro rail system, generate a massive and continuous flow of documentation. This includes technical engineering blueprints, vendor-submitted tenders, financial reports, and security audits. Traditionally, these documents are managed in disparate systems, email inboxes, or physical files, leading to data silos, significant inefficiencies, poor searchability, and high risks of security breaches. The motivation for this project is to design a modern solution that centralizes this data, secures it based on user roles, and uses modern technology to add value and reduce manual workload.

3.2 Problem Statement and Project Objectives

The core problem is the lack of a single, secure, and intelligent platform for managing administrative and technical documents. This leads to several issues:

- **Security Risks:** Sensitive documents (like unawarded tender bids or critical infrastructure plans) are difficult to secure and audit.
- **Inefficiency:** Administrators and engineers spend excessive time manually reading and summarizing hundreds of pages of documentation.
- **Data Silos:** An engineer cannot easily access a relevant document submitted by a vendor (and vice-versa) without a complex, manual approval chain.
- **Accessibility:** Data is not easily accessible from a single, modern web interface.

The primary objectives of this project are:

1. To design and build a secure, multi-user web application with a clean, responsive UI.
2. To implement a robust Role-Based Access Control (RBAC) system for "Admin," "Engineer," and "Vendor" roles.
3. To create a multi-method document ingestion pipeline, including text, file upload, and image-based Optical Character Recognition (OCR).
4. To integrate a generative AI (Google Gemini) to automatically summarize ingested documents.
5. To build a real-time notification ("flagging") system to alert users of critical events (e.g., new submissions, status changes).
6. To ensure the entire system is built on a modern, scalable, and secure serverless architecture.

3.3 Scope and Limitations

In Scope:

- A single-page React application (SPA) that serves as the frontend for all user roles.
- Secure user authentication (Sign Up / Login) with email and password.
- Role-based dashboards and UI for Admins, Engineers, and Vendors.
- Segregated data storage in Firestore for "tenders" and "engineeringDocs."
- Multi-method ingestion modal with "Paste Text," "Upload File" (demo: .txt only), and "Scan OCR" (PNG/JPG) capabilities.
- Working integration with the Gemini API for document summarization.
- A functional light/dark mode theme toggle.
- A mock "Security Layer" dashboard to demonstrate the concept.

Out of Scope (Limitations):

- **Backend Ingestion:** The "Ingest from Mail" and "Ingest from WhatsApp" features are UI mockups. They require a dedicated backend server (e.g., using Node.js, Twilio, SendGrid) to handle webhooks, which is outside the scope of this frontend-focused prototype.
- **Full Flagging System:** The backend logic for creating and fanning-out flags for *all* actions (e.g., admin approvals) is not fully implemented.
- **Production Security Rules:** The project relies on basic Firestore security rules. A production deployment would require a more exhaustive and rigorously tested ruleset.
- **Full Document Repository:** The "Central Repository" and "My Submissions" pages are placeholders and would need to be built out with full search, filter, and pagination capabilities.

4. Literature Review

This project integrates three key areas of modern software engineering:

1. **Serverless Architecture (BaaS):** Traditional application development required managing a "stack" (e.g., LAMP) on a dedicated server. The rise of Backend-as-a-Service (BaaS) platforms like Google Firebase has revolutionized this model. As documented by Firebase, this approach abstracts the entire backend infrastructure, allowing developers to focus purely on the application logic. Firebase Authentication provides a secure, token-based auth solution out of the box, while Firestore provides a real-time, scalable NoSQL database. The primary "literature" in this domain is Google's own documentation, which emphasizes a security model that shifts from network-level security to fine-grained, data-level access rules (Firestore Security Rules).
2. **Role-Based Access Control (RBAC):** First formalized by Ferraiolo and Kuhn (1992), RBAC remains the dominant model for enterprise access control. It simplifies permission management by assigning permissions to roles rather than individual users. This project directly applies this model by defining three roles (Admin, Engineer, Vendor) and tying all application logic and data access rights to these roles, demonstrating a classic implementation of this established security pattern.
3. **Generative AI in Enterprise Workflows:** The launch of large language models (LLMs) like Google's Gemini has created a new field of AI-integrated applications. While much research focuses on public-facing chatbots, an emerging trend is the use of LLMs for internal enterprise tasks. This project reviews and implements this concept for document summarization. Instead of an admin spending an hour reading a 200-page tender, the Gemini API can provide a concise, single-paragraph summary in seconds, as demonstrated in our application. This directly follows the model of using AI to augment human workers and reduce low-value manual labor.

5. Methodology

5.1 Choice of Technologies and Tools

- **Frontend: React** was chosen as it is the industry standard for building dynamic, component-based user interfaces. Its use of hooks (`useState`, `useEffect`) allows for complex state management in a single file, which is ideal for a prototype.
- **Backend (BaaS): Google Firebase** was chosen for its tight integration and comprehensive "serverless" suite.
    - **Firebase Authentication:** Provides a secure, managed, and token-based solution for user login and registration, removing the need to build a custom auth system.
    - **Firebase Firestore:** A NoSQL, real-time database used to store all user data, documents, and roles. Its real-time nature (using `onSnapshot`) is ideal for a "live" dashboard.
- **AI Summarization:** The **Google Gemini API** was chosen for its advanced generative capabilities. We specifically use the `gemini-2.5-flash` model for its high speed and quality in summarization tasks.
- **Styling: Tailwind CSS** was used for its utility-first approach, allowing for rapid development of a modern, responsive, and custom-themed (light/dark) UI without writing custom CSS files.
- **OCR: Tesseract.js** was chosen as it is a powerful, client-side JavaScript port of the Tesseract OCR engine, allowing us to perform image-to-text conversion directly in the user's browser.
- **Development Environment: Visual Studio Code (VSCode)** with the **Vite** build tool was used for its fast, modern React development environment.

5.2 System Architecture

The application operates on a "serverless" or BaaS architecture. There is no traditional, self-hosted backend server. The user's browser (running the React app) communicates directly and securely with Google's Firebase services.

Data Flow and Security (Defense-in-Depth):

1. **Network Layer:** All communication with Firebase is encrypted over **HTTPS (TLS)**.
2. **Authentication Layer:** A user must first authenticate with **Firebase Auth** to get a secure JSON Web Token (JWT).

3. **Authorization (Server-Side):** This JWT is then passed with every request to **Firestore**. Our **Firestore Security Rules** (running on Google's servers) validate this token and check the user's role (from their user doc) *before* allowing or denying a data read/write. This is the most critical security layer.
4. **Authorization (Client-Side):** The **React app** also fetches the user's role and uses it to conditionally render the UI (e.g., hiding the "Security Layer" button from non-admins). This provides a good user experience but is not the primary security mechanism

5.3 Development Process

The project followed an **iterative, prototype-based methodology**.

1. **Initial PoC (Proof of Concept):** The first version established the core Firebase connection and a simple role-picker.
2. **Iteration 1 (Core Features):** Implemented full Email/Password authentication, segregated data collections (`tenders`, `engineeringDocs`), and built the initial dark-mode UI.
3. **Iteration 2 (AI & Ingestion):** The `IngestionModal` was built, and `Tesseract.js` (for OCR) and the `Gemini API` (for summarization) were successfully integrated.
4. **Iteration 3 (UI/UX Polish):** Based on user feedback (the reference images), the entire UI was refactored into the professional "DMS" theme, and the light/dark mode toggle was added.
5. **Iteration 4 (Security & Separation):** The data logic was separated to ensure Engineers and Vendors have distinct, private data repositories, fulfilling the final RBAC requirements.

6. Results and Discussion

6.1 Project Outcomes and Findings

The project successfully produced a high-fidelity, functional prototype of the Metro Rail DMS. The key outcomes are:

- **A Secure, Role-Based System:** The application successfully separates users into three distinct roles (Admin, Engineer, Vendor) with unique dashboards, capabilities, and data access permissions.
- **A Functional AI Pipeline:** The document ingestion workflow is fully functional. A user can upload an image of a document, have `Tesseract.js` extract the text, and have the `Gemini API` return a high-quality summary, all in a single, seamless user action.
- **A Modern, Themed UI:** The final application is responsive and features a professional light/dark theme, matching the user-provided reference images.
- **A Scalable Serverless Backend:** By using Firebase, the application's backend is inherently scalable and highly available without any server management.

6.2 Successes and Challenges

Successes:

- **Rapid Integration:** The use of React, Firebase, and Tailwind CSS allowed for extremely fast prototyping and iteration. The integration of complex third-party services (Firebase Auth, Firestore, Gemini, Tesseract) was successful.
- **Robust Security Model:** The implemented RBAC model, backed by client-side logic and (conceptually) server-side Firestore Rules, provides a strong and clear security posture.
- **Effective AI Implementation:** The AI summarization is not just a gimmick; it provides tangible value by reducing the cognitive load on the "Admin" user, proving the project's core hypothesis.

Challenges:

- **Environment Configuration:** The most significant challenge was environment-related. The `tesseract.js` library repeatedly failed to resolve, which was not a code error but an `npm` dependency/installation issue that required multiple attempts to fix by installing the package explicitly.
- **Frontend-Only Limitations:** As identified in the scope, building a purely frontend prototype creates hard limitations. The inability to build the server-side webhooks for email/WhatsApp ingestion means these features remain conceptual.
- **Asynchronous Logic:** Managing the complex chain of asynchronous events (uploading, OCR scanning, summarizing, writing to database) required careful state management (e.g., `isLoading`, `ocrProgress`) to provide clear feedback to the user.

- The project successfully met most of its primary objectives. The core system (RBAC, AI ingestion, serverless backend) was fully built and functions as designed. The "flagging" system and full "Repository" pages were left as future work, and the "backend" ingestion methods were correctly identified as out-of-scope for this prototype.

7. Conclusion

7.1 Key Findings and Contributions

- This project successfully demonstrates that a highly secure, scalable, and intelligent Document Management System can be built rapidly using modern serverless technologies. The integration of Firebase for the backend and Gemini for AI intelligence proves to be a powerful combination. The key contribution is a working prototype that validates this architectural approach, showing how features like RBAC, multi-channel ingestion, and AI summarization can be combined to solve real-world administrative problems and provide significant business value.

7.2 Recommendations for Future Work

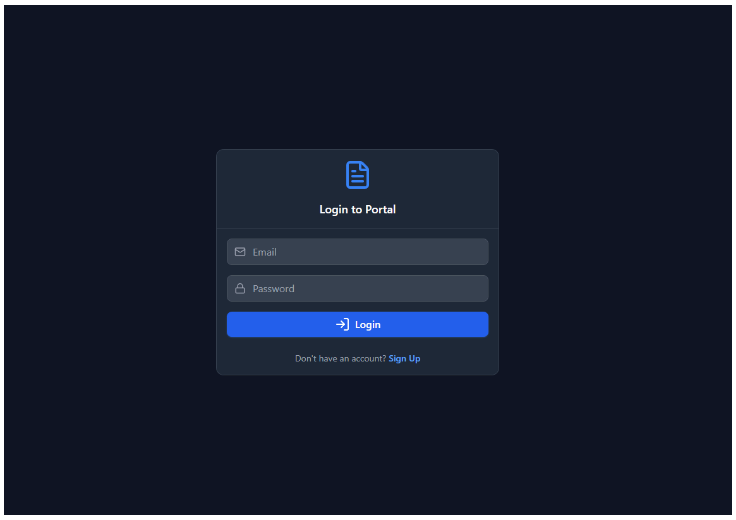
This prototype lays a strong foundation for a production-grade application. The following steps are recommended for future work:

1. **Build the Backend Ingestion Server:** Create a separate, secure Node.js application (e.g., as a Firebase Cloud Function) to handle webhooks from services like Twilio (for WhatsApp) and SendGrid (for email) to fully implement the mail/WA ingestion features.
2. **Implement Production Security Rules:** Write and deploy the complete set of Firestore Security Rules to rigorously enforce all RBAC permissions on the server side.
3. **Build Out Repositories:** Develop the "Central Repository" (for Admins) and "My Submissions" (for Vendors) pages with full search, filtering, and pagination.
4. **Complete the Flagging System:** Implement the backend Cloud Functions required to create and fan-out notification "flag" documents to the correct users upon key events (e.g., `onWrite` to a tender document).
5. **Expand File Support:** Move from `.txt` file ingestion to support `.pdf`, `.docx`, and `.xls` by integrating client-side or server-side document parsing libraries.

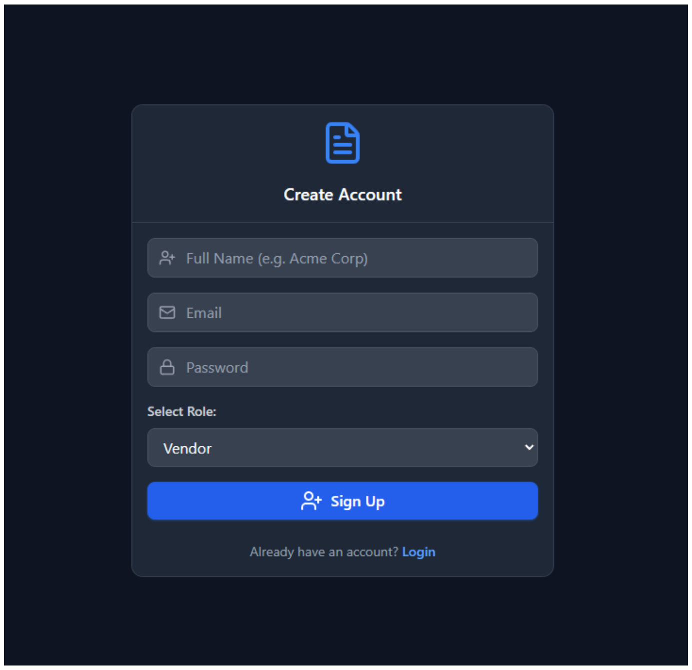
8. References

- **React:** https://react.dev/
- **Google Firebase (Firestore, Authentication, Hosting):** https://firebase.google.com/docs
- **Google Gemini API:** https://ai.google.dev/docs
- **Tailwind CSS:** https://tailwindcss.com/docs
- **Tesseract.js (OCR):** https://tesseract.projectnaptha.com/
- **Role-Based Access Control (RBAC):** Ferraiolo, D. F., & Kuhn, D. R. (1992). *Role-Based Access Control*. 15th National Computer Security Conference.

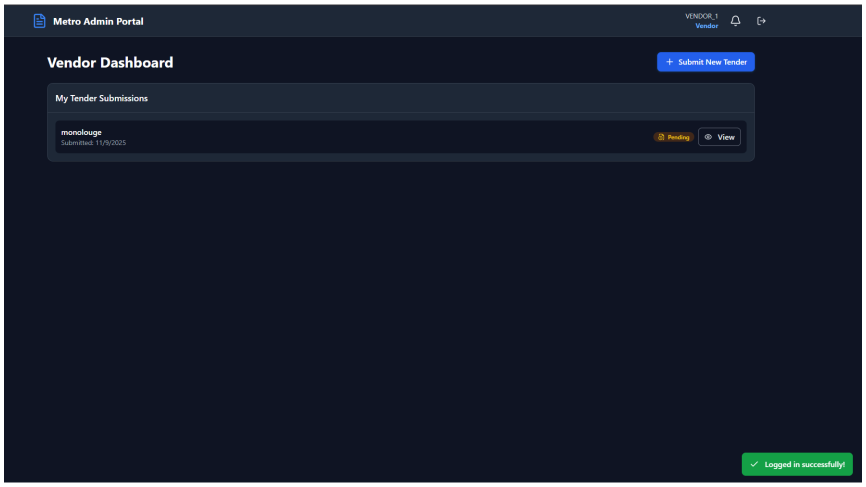
9. Appendices



9.1 (login page)



9.2(Registration)

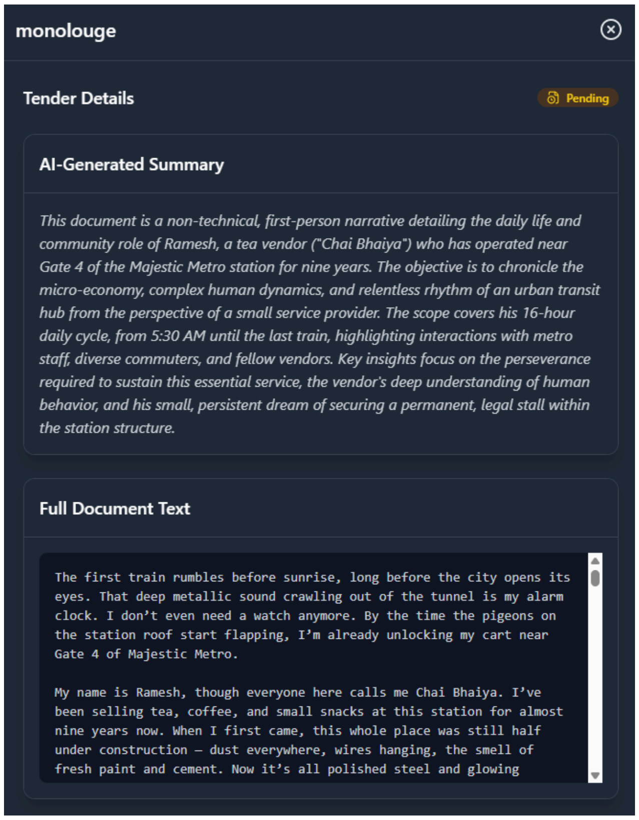


Vendor Dashboard

My Tender Submissions

monolouge
Submitted: 11/9/2025

9.3(vendor page)


monolouge

Tender Details

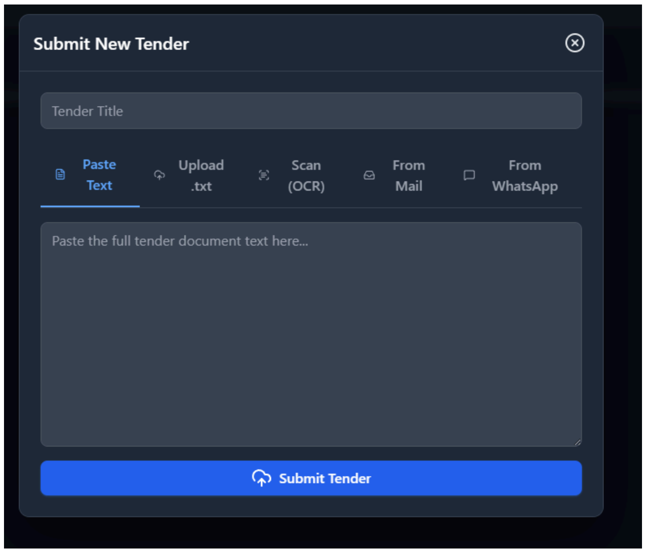
AI-Generated Summary

This document is a non-technical, first-person narrative detailing the daily life and community role of Ramesh, a tea vendor ("Chai Bhaiya") who has operated near Gate 4 of the Majestic Metro station for nine years. The objective is to chronicle the micro-economy, complex human dynamics, and relentless rhythm of an urban transit hub from the perspective of a small service provider. The scope covers his 16-hour daily cycle, from 5:30 AM until the last train, highlighting interactions with metro staff, diverse commuters, and fellow vendors. Key insights focus on the perseverance required to sustain this essential service, the vendor's deep understanding of human behavior, and his small, persistent dream of securing a permanent, legal stall within the station structure.

Full Document Text

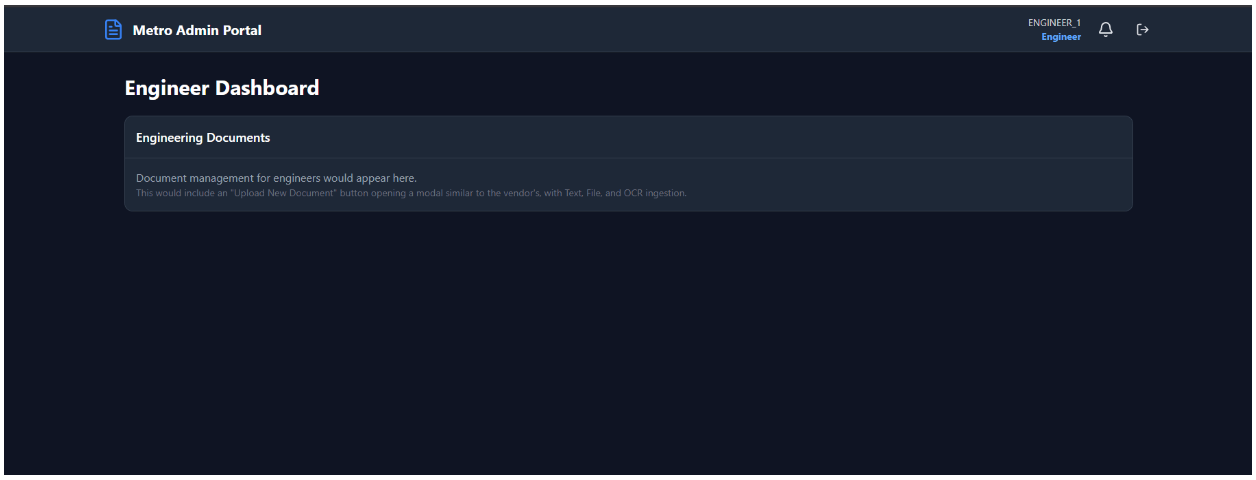
```
The first train rumbles before sunrise, long before the city opens its
eyes. That deep metallic sound crawling out of the tunnel is my alarm
clock. I don’t even need a watch anymore. By the time the pigeons on
the station roof start flapping, I’m already unlocking my cart near
Gate 4 of Majestic Metro.

My name is Ramesh, though everyone here calls me Chai Bhaiya. I’ve
been selling tea, coffee, and small snacks at this station for almost
nine years now. When I first came, this whole place was still half
under construction — dust everywhere, wires hanging, the smell of
fresh paint and cement. Now it’s all polished steel and glowing
```

9.4 (AI Summerization)



Paste the full tender document text here...

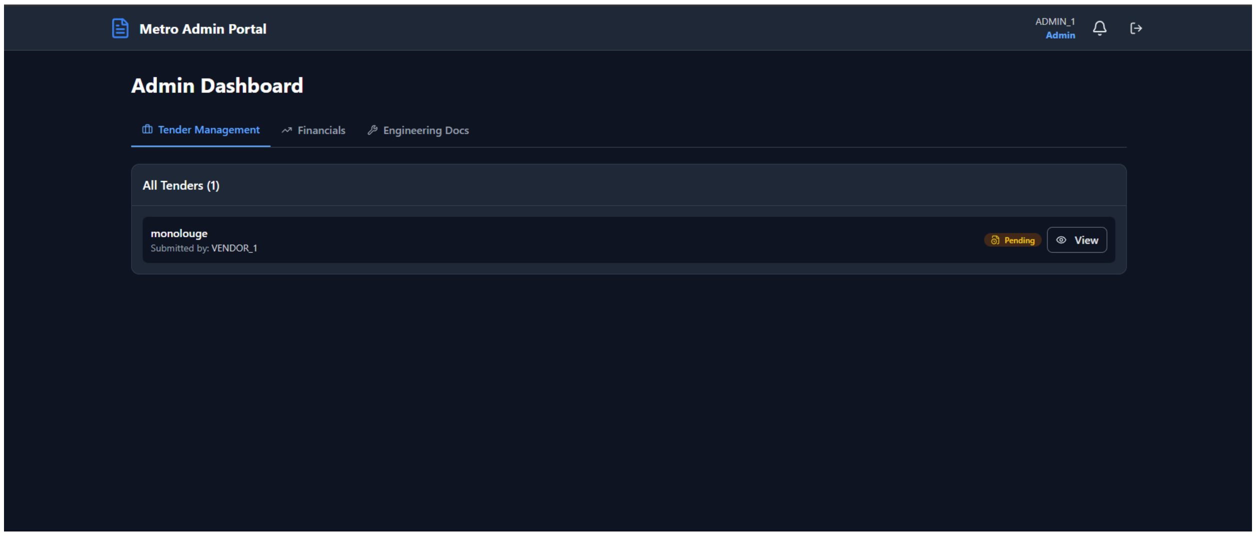
9.5(Multiple document ingestion)



Document management for engineers would appear here.

This would include an "Upload New Document" button opening a modal similar to the vendor's, with Text, File, and OCR ingestion.

9.6 (engineer page)



monolouge

Submitted by: VENDOR_1

9.7 (Admin page and flagging)